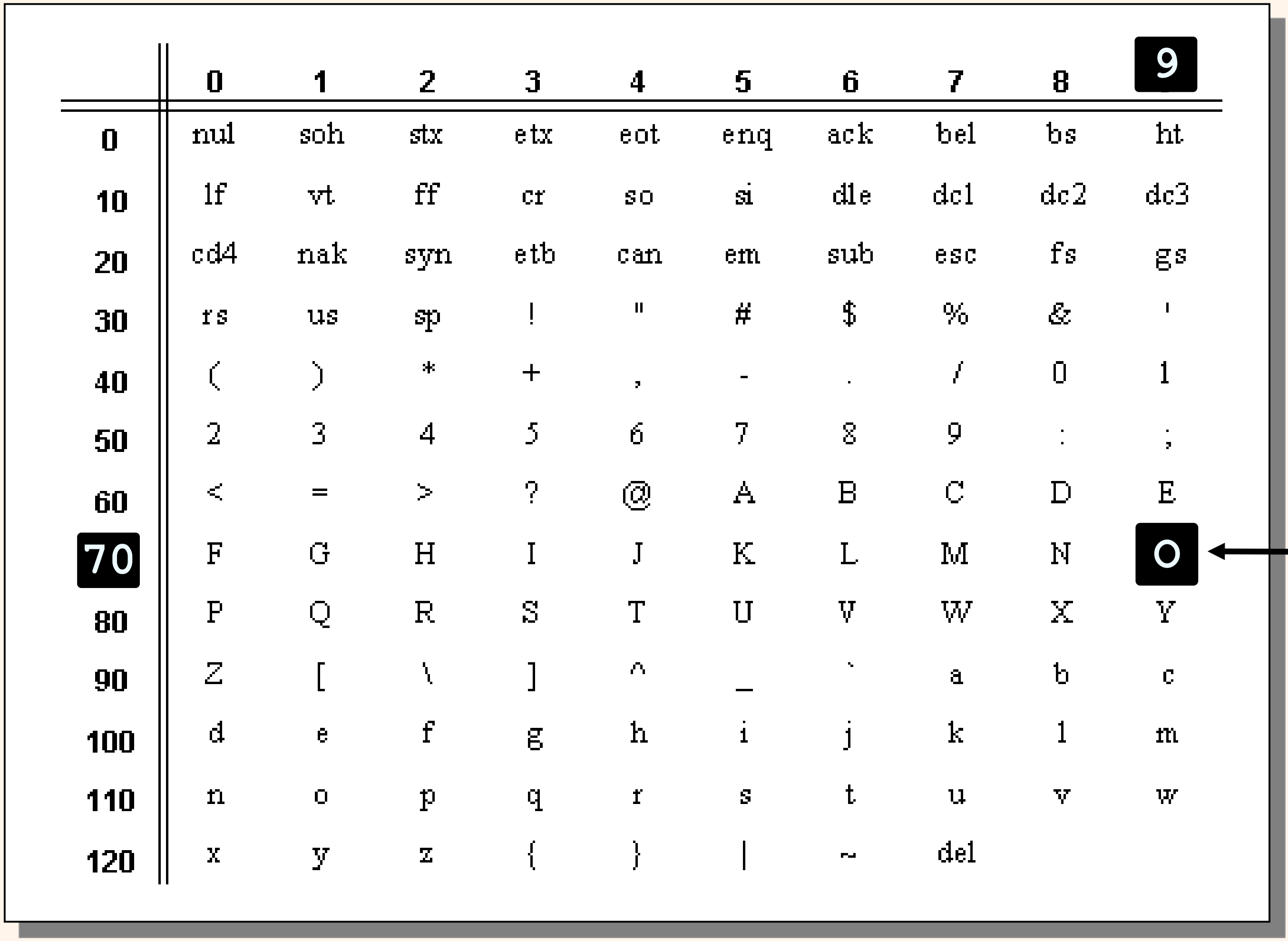
String and Characters

Characters

- In Java, single characters are represented using the data type **char**.
- Character constants are written as symbols enclosed in single quotes.
- Characters are stored in a computer memory using some form of encoding.
- *ASCII*, which stands for *American Standard Code for Information Interchange*, is one of the document coding schemes widely used today.
- Java uses Unicode, which includes ASCII, for representing **char** constants.

ASCII Encoding

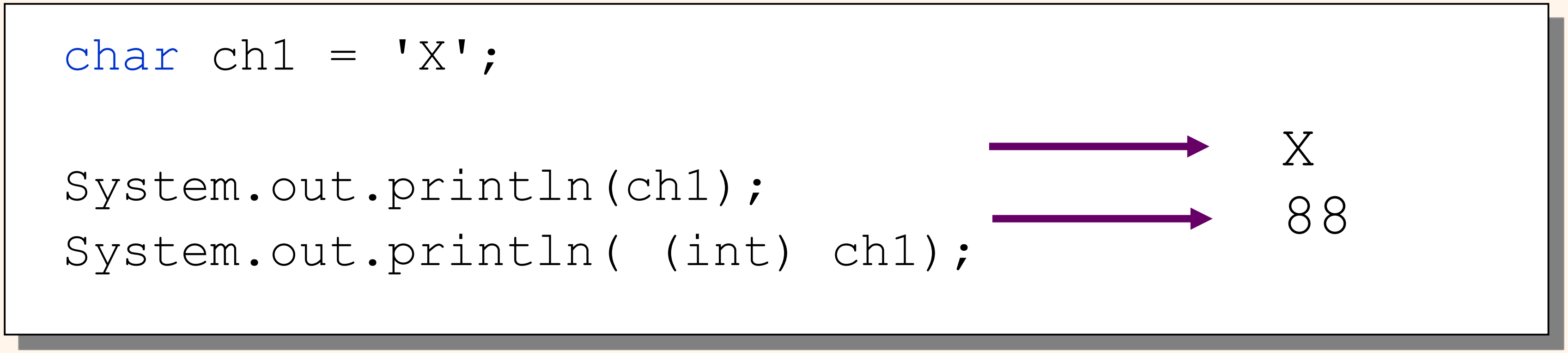


	0	1	2	3	4	5	6	7	8	9
0	nul	soh	stx	etx	eot	enq	ack	bel	bs	ht
10	lf	vt	ff	cr	so	si	dle	dc1	dc2	dc3
20	cd4	nak	syn	etb	can	em	sub	esc	fs	gs
30	rs	us	sp	!	"	#	$	%	&	'
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9	:	;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	^	_	`	a	b	c
100	d	e	f	g	h	i	j	k	l	m
110	n	o	p	q	r	s	t	u	v	w
120	x	y	z	{	}	|	~	del		

For example, character 'O' is 79 (row value 70 + col value 9 = 79).

Unicode Encoding

- The *Unicode Worldwide Character Standard* (*Unicode*) supports the interchange, processing, and display of the written texts of diverse languages.
- Java uses the Unicode standard for representing **char** constants.



```
char ch1 = 'X';

System.out.println(ch1);
System.out.println( (int) ch1);
```

```
X
88
```

Character Processing

```
char ch1, ch2 = ‘X’;
```

Declaration and initialization

```
System.out.print("ASCII code of character X is " +
                 (int) 'X' );

System.out.print("Character with ASCII code 88 is "
         + (char)88 );
```

Type conversion between int and char.

```
‘A’ < ‘c’
```

This comparison returns true because ASCII value of 'A' is 65 while that of 'c' is 99.

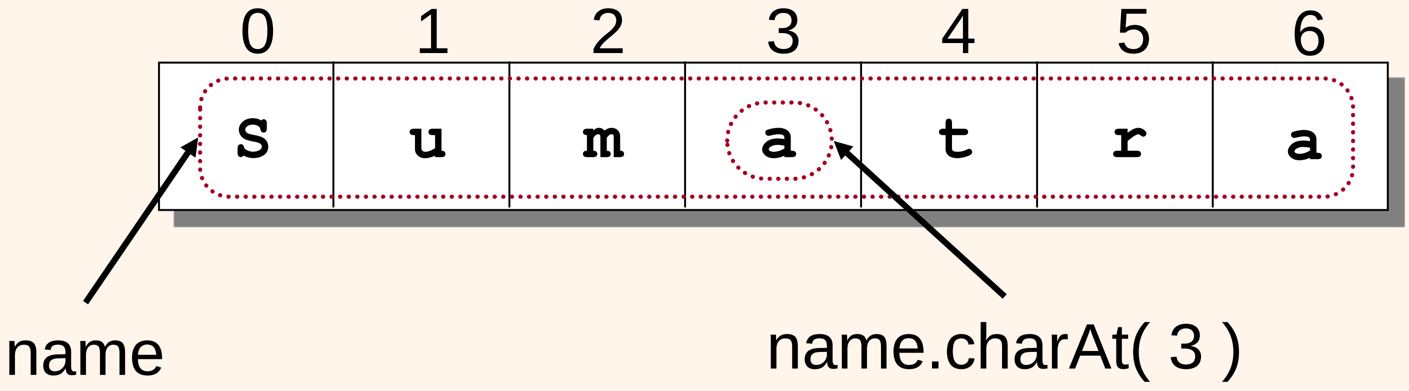
Strings

- A *string* is a sequence of characters that is treated as a single value.
- Instances of the **String** class are used to represent strings in Java.
- We can access individual characters of a string by calling the **charAt** method of the **String** object.

Accessing Individual Elements

- Individual characters in a String accessed with the charAt method.

```
String name = "Sumatra";
```



This variable refers to the whole string.

The method returns the character at position # 3.

Example: Counting Vowels

```java
char     letter;
System.out.println("Your name:");
String   name = scanner.next(); //assume ‘scanner’ is created properly
int      numberOfCharacters = name.length();
int      vowelCount   = 0;

for (int i = 0; i < numberOfCharacters; i++) {
    letter = name.charAt(i);

    if (   letter == 'a' || letter == 'A' ||
           letter == 'e' || letter == 'E' ||
           letter == 'i' || letter == 'I' ||
           letter == 'o' || letter == 'O' ||
           letter == 'u' || letter == 'U'
                                          ) {
         vowelCount++;
    }
}
System.out.print(name + ", your name has " + vowelCount + " vowels");
```

Here's the code to count the number of vowels in the input string.

Example: Counting ‘Java’

```
int          javaCount      = 0;
boolean      repeat         = true;
String       word;
Scanner       scanner = new Scanner(System.in);

while ( repeat ) {
     System.out.print("Next word:");
     word = scanner.next();

     if ( word.equals("STOP") )      {
           repeat = false;

     } else if ( word.equalsIgnoreCase("Java") ) {
           javaCount++;
     }
}
```

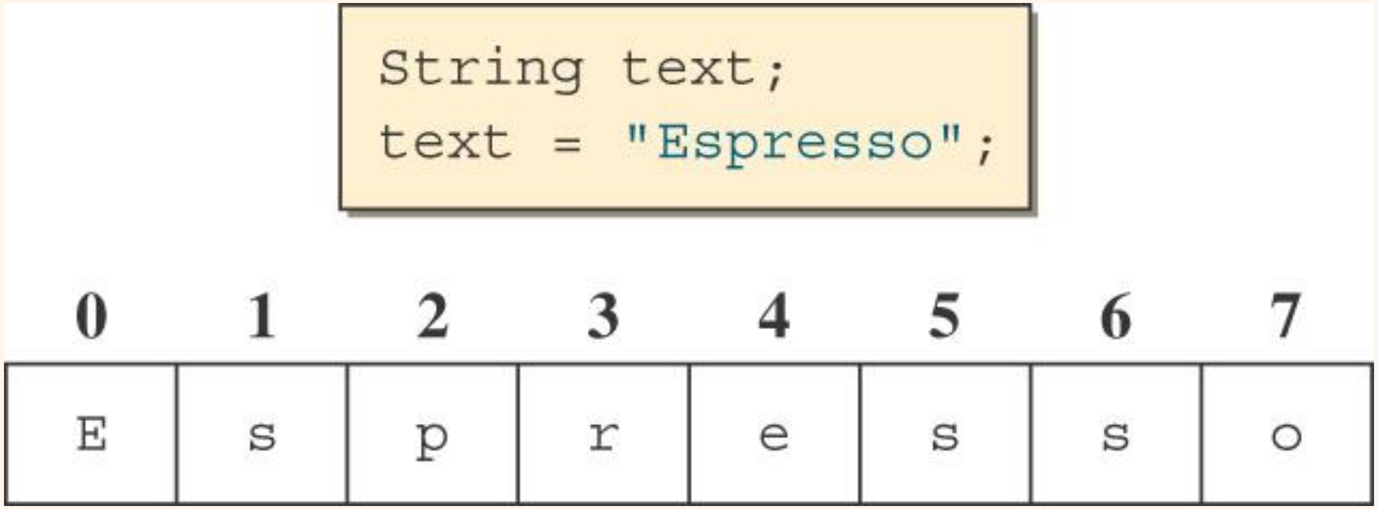
Continue reading words and count how many times the word Java occurs in the input, ignoring the case.

Notice how the comparison is done. We are not using the == operator.

String

- The textual values passed to the showMessageDialog method are instances of the String class.
- A sequence of characters separated by double quotes is a String constant.
- There are close to 50 methods defined in the String class. We will introduce three of them here: substring, length, and indexOf.
- We will also introduce a string operation called concatenation.

String Indexing



```
String text;
text = "Espresso";
```

0	1	2	3	4	5	6	7
E	s	p	r	e	s	s	o

The position, or index, of the first character is 0.

Definition: substring

- Assume str is a String object and properly initialized to a string.
- str.substring(i, j) will return a new string by extracting characters of str from position i to j-1 where $0 \leq i < \text{length of str}$, $0 < j \leq \text{length of str}$, and $i \leq j$.
- If str is “programming” , then str.substring(3, 7) will create a new string whose value is “gram” because g is at position 3 and m is at position 6.
- The original string str remains unchanged.

Examples: substring

```
String text = “Espresso”;
```

`text.substring(6,8)`	→	“so”
`text.substring(0,8)`	→	“Espresso”
`text.substring(1,5)`	→	“spre”
`text.substring(3,3)`	→	“”
`text.substring(4,2)`	→	error

Definition: length

- Assume str is a String object and properly initialized to a string.
- str.length( ) will return the number of characters in str.
- If str is "programming" , then str.length( ) will return 11 because there are 11 characters in it.
- The original string str remains unchanged.

Examples: length

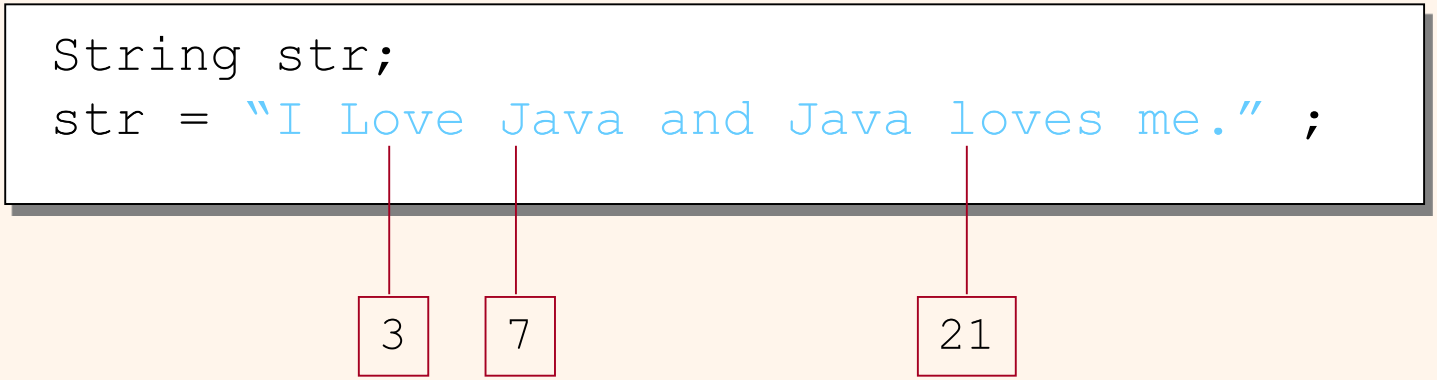
```
String str1, str2, str3, str4;
str1 = “Hello” ;
str2 = “Java” ;
str3 = “” ; //empty string
str4 = “ “ ; //one space
```

```
str1.length( )   →   5

str2.length( )   →   4

str3.length( )   →   0

str4.length( )   →   1
```

Definition: indexOf

- Assume str and substr are String objects and properly initialized.
- str.indexOf(substr) will return the first position substr occurs in str.
- If str is “programming” and substr is “gram” , then str.indexOf(substr) will return 3 because the position of the first character of substr in str is 3.
- If substr does not occur in str, then –1 is returned.
- The search is case-sensitive.

Examples: indexOf



```
String str;
str = "I Love Java and Java loves me." ;
```

```
str.indexOf( "J" )          → 7
str2.indexOf( "love" )      → 21
str3. indexOf( "ove" )      → 3
str4. indexOf( "Me" )       → -1
```

Definition: concatenation

- Assume str1 and str2 are String objects and properly initialized.
- str1 + str2 will return a new string that is a concatenation of two strings.
- If str1 is “pro” and str2 is “gram” , then str1 + str2 will return “program”.
- Notice that this is an operator and not a method of the String class.
- The strings str1 and str2 remains the same.

Examples: concatenation

```
String str1, str2;
str1 = “Jon” ;
str2 = “Java” ;
```

```
str1 + str2                     →  “JonJava”

str1 + “ “ + str2               →  “Jon Java”

str2 + “, “ + str1              →  “Java, Jon”

“Are you “ + str1 + “?”         →  “Are you Jon?”
```

Comparing Objects

- With primitive data types, we have only one way to compare them, but with objects (reference data type), we have two ways to compare them.
    1. We can test whether two variables point to the same object (use ==), or
    2. We can test whether two distinct objects have the same contents.

Using == With Objects (Sample 1)

```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are not equal
```

Not equal because str1 and str2 point to different String objects.

Using == With Objects (Sample 2)

```
String str1 = new String("Java");
String str2 = str1;

if (str1 == str2) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 point to the same object.

Using equals with String

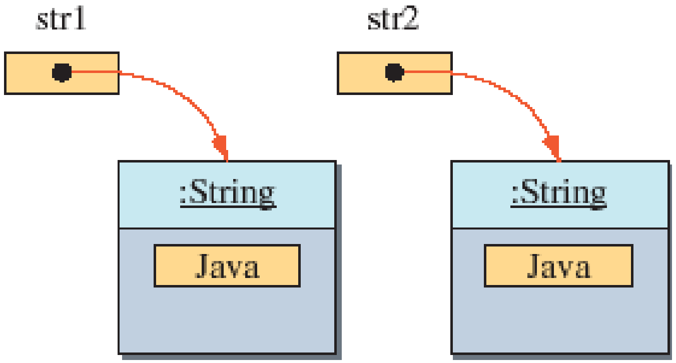
```
String str1 = new String("Java");
String str2 = new String("Java");

if (str1.equals(str2)) {
    System.out.println("They are equal");
} else {
    System.out.println("They are not equal");
}
```

```
They are equal
```

It's equal here because str1 and str2 have the same sequence of characters.

The Semantics of ==

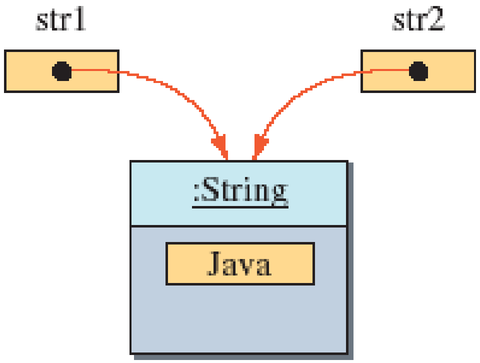
Case A: Two variables refer to two different objects.



```
String str1, str2;

str1 = new String("Java");
str2 = new String("Java");
```

str1 == str2 → false

Case B: Two variables refer to the same object.



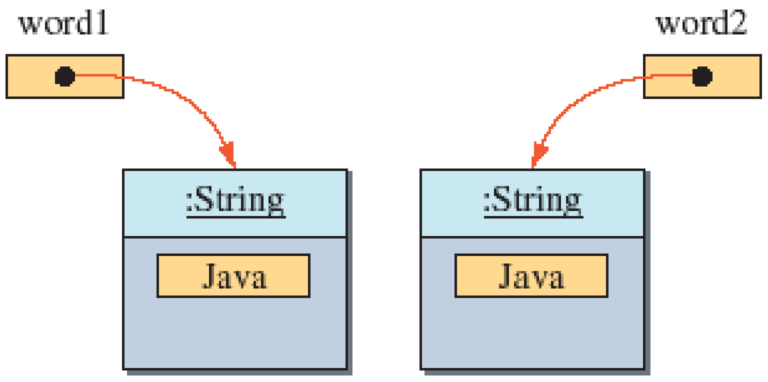
```
String str1, str2;

str1 = new String("Java");
str2 = str1;
```

str1 == str2 → true

In Creating String Objects

```
String word1, word2;

word1 = new String("Java");

word2 = new String("Java");
```

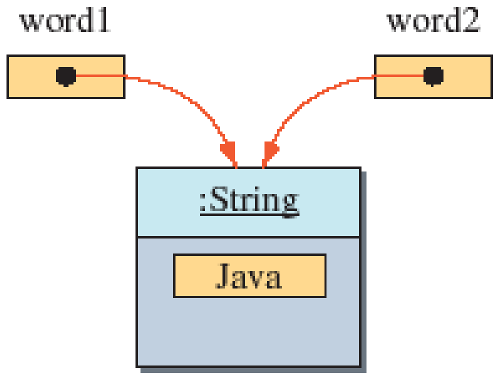
Whenever the new operator is used, there will be a new object.



word1 == word2 → false

```
String word1, word2;

word1 = "Java";

word2 = "Java";
```

Literal string constant such as “Java” will always refer to one object.



word1 == word2 → true

Other Useful String Operators

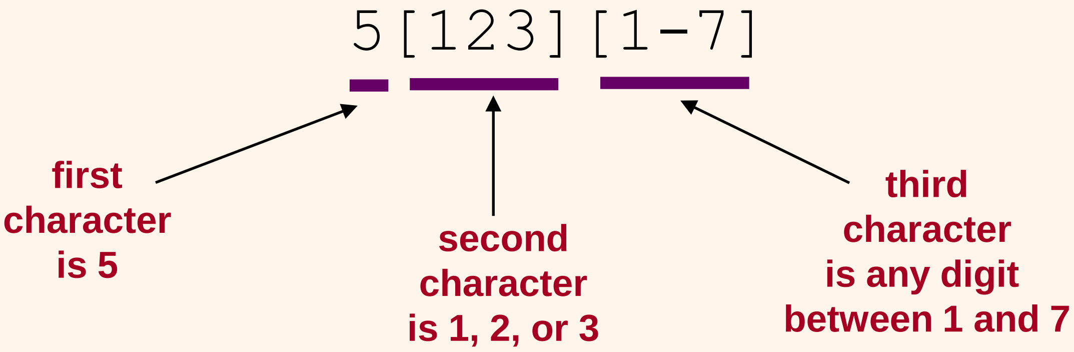
Method	Meaning
compareTo	Compares the two strings. `str1.compareTo( str2 )`
substring	Extracts the a substring from a string. `str1.substring( 1, 4 )`
trim	Removes the leading and trailing spaces. `str1.trim( )`
valueOf	Converts a given primitive data value to a string. `String.valueOf( 123.4565 )`
startsWith	Returns true if a string starts with a specified prefix string. `str1.startsWith( str2 )`
endsWith	Returns true if a string ends with a specified suffix string. `str1.endsWith( str2 )`

- See the String class documentation for details.

Pattern Example

- Suppose students are assigned a three-digit code:
  - The first digit represents the major (5 indicates computer science);
  - The second digit represents either in-state (1), out-of-state (2), or foreign (3);
  - The third digit indicates campus housing:
    - On-campus dorms are numbered 1-7.
    - Students living off-campus are represented by the digit 8.

The 3-digit pattern to represent computer science majors living on-campus is



```
5[123][1-7]
```

first character is 5

second character is 1, 2, or 3

third character is any digit between 1 and 7

Regular Expressions

- The pattern is called a *regular expression*.
- Rules
  - The brackets [ ] represent choices
  - The asterisk symbol * means zero or more occurrences.
  - The plus symbol + means one or more occurrences.
  - The hat symbol ^ means negation.
  - The hyphen – means ranges.
  - The parentheses ( ) and the vertical bar | mean a range of choices for multiple characters.

Regular Expression Examples

Expression	Description
`[013]`	A single digit 0, 1, or 3.
`[0-9][0-9]`	Any two-digit number from 00 to 99.
`[0-9&&[^4567]]`	A single digit that is 0, 1, 2, 3, 8, or 9.
`[a-z0-9]`	A single character that is either a lowercase letter or a digit.
`[a-zA-z][a-zA-Z0-9_$]*`	A valid Java identifier consisting of alphanumeric characters, underscores, and dollar signs, with the first character being an alphabet.
`[wb](ad|eed)`	Matches `wad`, `weed`, `bad`, and `beed`.
`(AZ|CA|CO)[0-9][0-9]`	Matches `AZxx`,`CAxx`, and `COxx`, where `x` is a single digit.

The replaceAll Method

- The replaceAll method replaces all occurrences of a substring that matches a given regular expression with a given replacement string.

Replace all vowels with the symbol @

```
String originalText, modifiedText;

originalText = ...;    //assign string

modifiedText =
         originalText.replaceAll("[aeiou]","@");
```

The Pattern and Matcher Classes

- The matches and replaceAll methods of the String class are shorthand for using the Pattern and Matcher classes from the java.util.regex package.
- If str and regex are String objects, then

```
str.matches(regex);
```

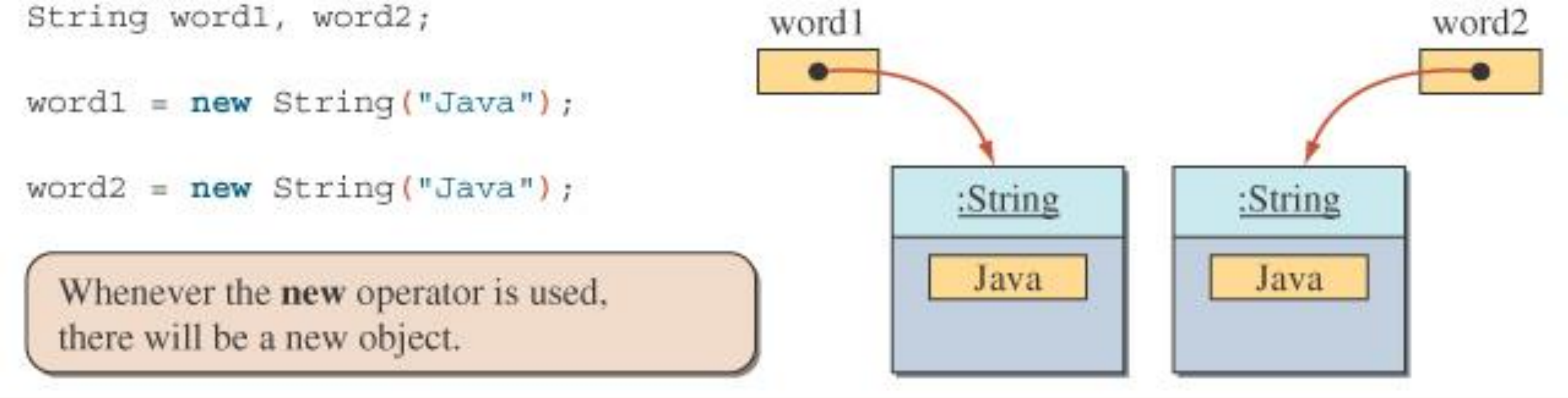
is equivalent to

```
Pattern pattern = Pattern.compile(regex);
Matcher matcher = pattern.matcher(str);
matcher.matches();
```

The String Class is Immutable

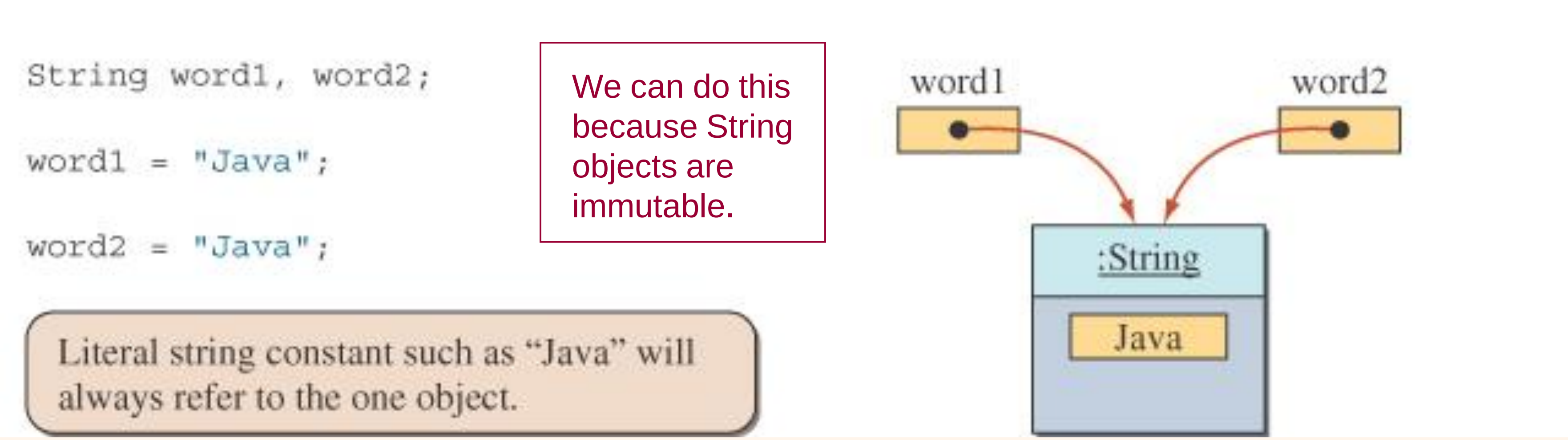
- In Java a String object is immutable
  - This means once a String object is created, it cannot be changed, such as replacing a character with another character or removing a character
  - The String methods we have used so far do not change the original string. They created a new string from the original. For example, substring creates a new string from a given string.
- The String class is defined in this manner for efficiency reason.

Effect of Immutability



```
String word1, word2;

word1 = new String("Java");

word2 = new String("Java");
```

Whenever the **new** operator is used, there will be a new object.



```
String word1, word2;

word1 = "Java";

word2 = "Java";
```

We can do this because String objects are immutable.

Literal string constant such as “Java” will always refer to the one object.

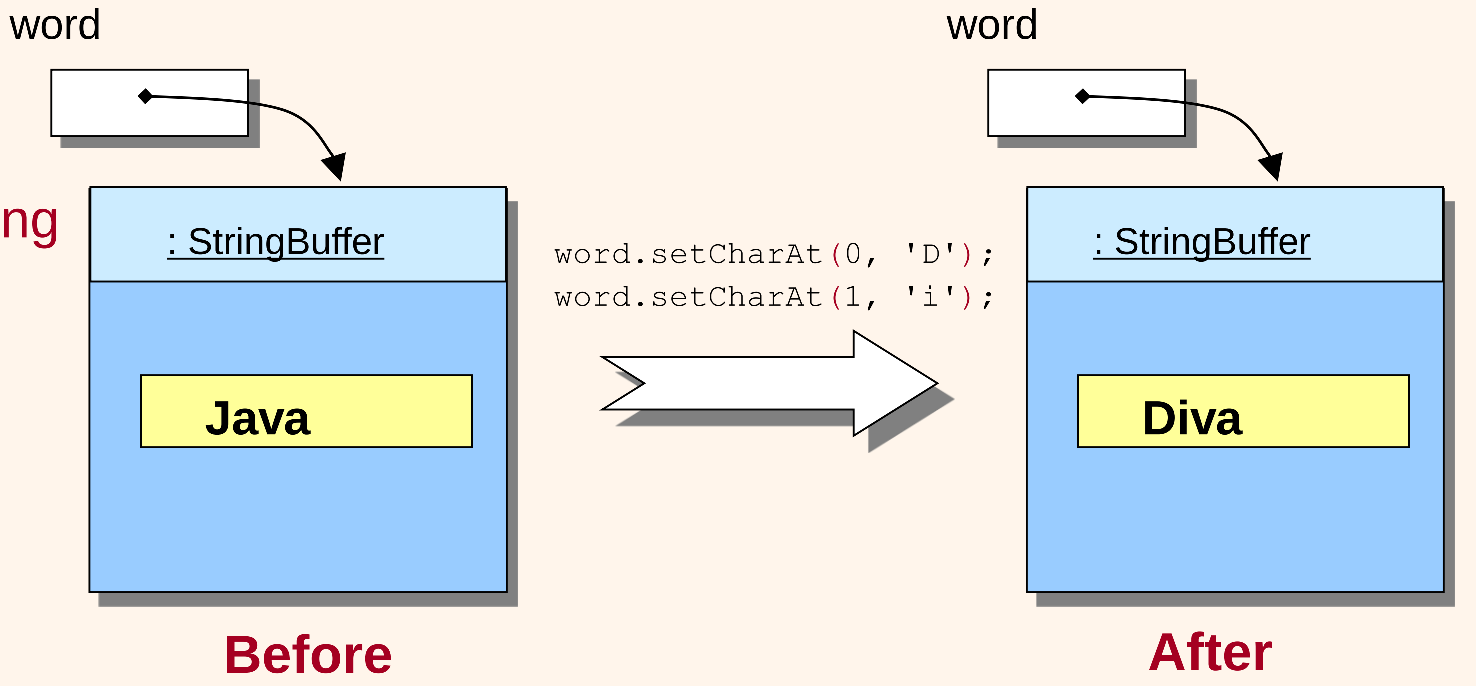
The StringBuffer Class

- In many string processing applications, we would like to change the contents of a string. In other words, we want it to be mutable.
- Manipulating the content of a string, such as replacing a character, appending a string with another string, deleting a portion of a string, and so on, may be accomplished by using the **StringBuffer** class.

StringBuffer Example

```
StringBuffer word = new StringBuffer("Java");
word.setCharAt(0, 'D');
word.setCharAt(1, 'i');
```

Changing a string
Java to Diva



```
word.setCharAt(0, 'D');
word.setCharAt(1, 'i');
```

Sample Processing

Replace all vowels in the sentence with ‘X’.

```
char          letter;
String        inSentence    = JOptionPane.showInputDialog(null, "Sentence:");
StringBuffer tempStringBuffer   = new StringBuffer(inSentence);
int           numberOfCharacters = tempStringBuffer.length();

for (int index = 0; index < numberOfCharacters; index++) {

    letter = tempStringBuffer.charAt(index);

    if (letter == 'a' || letter == 'A' || letter == 'e' || letter == 'E' ||
        letter == 'i' || letter == 'I' || letter == 'o' || letter == 'O' ||
        letter == 'u' || letter == 'U'  ) {
        tempStringBuffer.setCharAt(index,'X');
    }
}
JOptionPane.showMessageDialog(null, tempStringBuffer );
```

The append and insert Methods

- We use the append method to append a String or StringBuffer object to the end of a StringBuffer object.
  - The method can also take an argument of the primitive data type.
  - Any primitive data type argument is converted to a string before it is appended to a StringBuffer object.
- We can insert a string at a specified position by using the insert method.

```
import java.util.*;
public class question3 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        String s = "Hello, world!";
        StringBuilder t = new StringBuilder(s);
        t.setCharAt(1, 'i');
        t.delete(2, 5);
        System.out.println(t);
    }
}
```